title: Bundle Inclusion Troubleshooting

How to troubleshoot your Flashbots bundle not landing on-chain

Unlike broadcasting a transaction which lands on-chain even if the transaction fails, troubleshooting Flashbots bundles is considerably more challenging, since any of the following circumstances will prevent your bundle from landing on chain:

1. Any transaction failure within the bundle that isn't specified in the optional argument `revertingTxHashes` 2. Insufficient incentives (the sum of gas price and coinbase transfers) to compensate for the value of block space 3. Higher bids from competitors for the same opportunity 4. Late receipt of the bundle, preventing its inclusion in the target block 5. The validator for the target slot is not running mev-boost

Instead of relying on Etherscan to examine the execution of your transaction, its on-chain status, and its comparison with competitors, you'll need a different approach with Flashbots. This is because Flashbots prevents failed transactions from appearing on the chain. For effective debugging, we highly recommend simulating your transactions, logging the results, and maintaining a record of all submitted data, including the complete bundle and its signed transactions.

The issues listed above are arranged in the order of their priority for consideration. In the following sections, we will explore each issue in detail, providing guidance on how to identify and address them. While the examples provided assume the use of the Flashbots Ethers Provider, the RPC calls are standard and the suggested strategies can be easily adapted for use with other providers.

Does your transaction work and pay enough?

Covers:

1. Transaction failure (ANY within the bundle) 2. Incentives (gas price/coinbase transfers) not high enough to offset value of block space

The first two issues are grouped together as their causes, investigations, and solutions are closely related. Flashbots will not include a bundle if:

1. A transaction within the bundle reverts, unless it's specified via the optional argument revertingTxHashes or it's uncled.
2. The gas price is below the base fee, as this would result in an invalid block if included.
3. The effective priority fee is not sufficient to offset the opportunity cost of using the block space for other unrelated transactions. For instance, if your bundle is paying a 1 Gwei priority fee, but the cheapest transaction in the block is paying 2 Gwei, the builder would benefit more from discarding your bundle in favor of standard pending transactions.

As any of these conditions result in your bundle not appearing in a block, troubleshooting these issues requires *simulation* using eth_callBundle RPC call. eth_callBundle is similar to an eth_call you might already be familiar with, but offers these key benefits:

1. It operates on an array of *signed* transactions, as opposed to a single unsigned transaction description. These transactions are executed sequentially, starting from the top of the specified block. Simulating with signed transactions minimizes discrepancies between how your system creates transactions and how they will be processed on-chain. For instance, it eliminates the possibility of using an incorrect from field when simulating a signed transaction.
2. It provides the gas used and the coinbase transfer for each transaction. The coinbase transfer is a factor in the effective gas price calculation.
3. It allows for the precise specification of the following arguments, enabling a more accurate simulation:
4. State block number: This determines the values read from SLOADs.
5. EVM block number: This determines the value returned from block.number.
6. EVM timestamp: This determines the value returned from block.timestamp.

The Flashbots ethers.js provider exposes eth_callBundle via the simulate() method. This only operates on a pre-signed bundle, so you must sign your bundle transactions manually.

```
js const signedTransactions = await flashbotsProvider.signBundle(transactionBundle); const simulation = await flashbotsProvider.simulate( signedTransactions, targetBlockNumber, targetBlockNumber + 1, ); console.log(JSON.stringify(simulation, null, 2));
```

Output:

```
js { "totalGasUsed": 98564, "bundleHash": "0x9a6a9fa038343fe3c57260fb7bdb2c79ebadb3088656300d8a494123ebda6d85", "coinbaseDiff": BigNumber(0x034dc9949767a4), }, "results": [ { "coinbaseDiff": "929953106847652", "ethSentToCoinbase": "0", "fromAddress": "0x9874Ef8519a0Fc7a6B553aad92fDF0E469488931", "gasFees": "929953106847652", "gasPrice": "35008022393", "gasUsed": 29964, "toAddress": "0x48B2dD9CEFbA73c60882478a16BC3428Aceed2B9", "txHash": "0xee3f6f22bf3b4740b36833d41d4872f48f98c6328fa04b3679558e482ba0e328", "value": "0x0000000000000000000000000000000000000000000000000000000000000001" }, ... ], }
```

To resolve, ensure the response from eth_callBundle does not revert and matches your expectations for bundle profitability. Compare your bundle's effective gas price against the profit of the conflicting bundle.

Is your bundle paying enough to be competitive?

Covers:

3. Competitors paying more

Flashbots bundles adhere to a "blind" auction, where bundle pricing is not released by Flashbots prior to landing on-chain. The winning "bids" are revealed *only* after the block containing winning bundles is propagated (via the transactions themselves and blocks-api for recognizing which set of transactions belong to a bundle).

While you cannot see a competitor's bid in real time, it is possible to look AFTER the fact to:

1. Identify the exact bundle (if any) that conflicted with yours
2. Compare the conflicting bundle's effective priority fee with your own (to see if you should be bidding more to remain competitive)

Types of conflicts

There are numerous reasons why two bundles might conflict. Consider that the basic algorithm for merging bundles is:

1. Simulate each bundle at the top of the block
2. Sort bundles by effective priority fee, highest first
3. In descending order, try each bundle **that does not error or lower its effective priority fee from top-of-block simulation** until you reach a maximum bundle inclusion count or you run out of profitable bundles

A conflict occurs when a bundle simulates one way at the top of the block, and a different [worse] way when placed after another bundle. Here is a list of the ways a bundle could conflict:

1. **Nonce collision** - The target bundle includes a transaction from account A and nonce B. The conflicting bundle also includes a transaction from account A and nonce B. The most common case for nonce collision is from including the exact same transaction, but it doesn't have to be; the conflicting bundle only needs to increment an account's nonce via any transaction.
2. **Revert** - The target bundle has no reverting transactions when simulated at the top of the block, but reverts when placed after a conflicting bundle that appears first
3. **Effective Priority Fee** - A bundle cannot significantly reduce its priority fee between simulating at the top of the block and when it is selected for inclusion. This commonly occurs when a bundle is operating on an arbitrage for which it pays a % of the profit to the validator, with an earlier bundle taking part, but not all, of the arbitrage opportunity.

Detecting

If a block you targeted contained Flashbots bundles, but yours did not appear, the next step is to determine which bundles conflicted with yours and, if present, calculate their effective priority fee. This can be accomplished through several iterations of simulations, using this strategy:

1. Simulate your bundle at the head of the target block, note its revert states and effective priority fee
2. Fetch all bundles found in the target block
3. Simulate [bundle1 + your bundle] as a single bundle, check the behavior of your bundle
4. Simulate [bundle1 + bundle2 + your bundle] as a single bundle, see the behavior of your bundle
5. Simulate [bundle1 + bundle2 + ...bundleN + your bundle] as a single bundle, see the behavior of your bundle
6. And so on...

Using this method, we can identify the conflicting bundle that caused your target bundle to change behavior. The Flashbots ethers.js provider has a built-in helper function for running this strategy called getConflictingBundle():

```
js const signedTransactions = await flashbotsProvider.signBundle(transactionBundle); console.log( await flashbotsProvider.getConflictingBundle( signedTransactions, 13140328, // blockNumber ), );
```

Output:

```
js { "conflictType": FlashbotsBundleConflictType.NonceCollision, "initialSimulation": { "totalGasUsed": 205860, "bundleHash": "0x1720ea33d96dca026dddd5689f8cad21966988348ced04e9054a0dca5d60f1d4", "coinbaseDiff": BigNumber(0x0176750858d000), }, "results": [...] }, "targetBundleGasPricing": { "gasUsed": 205860, "txCount": 1, "gasFeesPaidBySearcher": BigNumber(0x0176750858d000), "priorityFeesReceivedByMiner": BigNumber(0x52efd8d80dbc24), "ethSentToCoinbase": BigNumber.from(0x00), "effectiveGasPriceToSearcher": BigNumber(0x77359400), "effectivePriorityFeeToMiner": BigNumber(0x1a6734f601) }, "conflictingBundleGasPricing": { "gasUsed": 396462, "txCount": 3, "gasFeesPaidBySearcher": BigNumber(0xc4c3c97ce1bff8b4), "priorityFeesReceivedByMiner": BigNumber(0xc4213e4d7ad82006), "ethSentToCoinbase": BigNumber(0xc4c2663d3b804731), "effectiveGasPriceToSearcher": BigNumber(0x410ce509aa1e), "effectivePriorityFeeToMiner": BigNumber(0x40f2069f201d) }, "conflictingBundle": [ { "transaction_hash": "0x23a33038289dda1b6e722835d2b9388cb41d96d085c19ca6b71bb3e9697e6692", "tx_index": 0, "bundle_type": "flashbots", "bundle_index": 0, "block_number": 13140328, "eoa_address": "0x38563699560e4512c7574C8cC5Cf89fd43923BcA", "to_address": "0x000000000035B5e5ad9019092C665357240f594e", "gas_used": 100893, "gas_price": "0", "coinbase_transfer": "0", "total_miner_reward": "0" }, ... ] }
```

:::note

Parameters with miner in the name are retrofitted with Flashbots block builder data to maintain backwards compatibility. This

nomenclature will be changed in a future release to accurately reflect PoS Ethereum architecture.

:::

To resolve, first determine if you have an issue of competitors paying more and, if so, increase your `effective priority fee`. This can be accomplished either by paying more to the builder or validator, *or* using less gas to accomplish the same opportunity.

If your bundles are not outbid by a conflicting bundle, check to see if your bundles are being received too late:

Is your bundle received too late?

Covers:

4. Bundle received too late to appear in target block

Each bundle submission is designed to target a specific block number. Therefore, it's crucial to ensure that your bundle is received as promptly as possible. This allows the bundle sufficient time to:

1. Arrive at the builder
2. Undergo simulation
3. Be included in a block
4. Reach the validator through an mev-boost relay

All these steps need to be completed before the targeted block is proposed. If you're targeting `blockNumber +1`, which is common for most bundles, it's vital to deliver your bundle to your builder(s) as quickly as you can.

It's important to remember that there's a time frame for every block when your local perspective of block height is `X`, while `X+1` has already been discovered and propagated to a portion of the network, but hasn't reached your local node yet. During this period of partial propagation, submitting a bundle targeting `X+1` may seem valid from your network perspective, but could be pointless if builders have already started working on solving X+2. In rare cases, targeting the `X+1` block just before its discovery can also cause bundle failure due to insufficient time for the bundle to complete the above listed 4 steps, each of which takes aroun 1-2 seconds.

To monitor the time taken from the submission of your bundle to the Flashbots and the proposal of the next block, Flashbots provides an RPC endpoint `eth_getBundleStats`. This endpoint returns timing information based on a previously-submitted bundle. Each submitted bundle is uniquely identified by a bundleHash and target block number for future reference. The bundleHash is straightforward to calculate, as shown here.

```
js console.log( await flashbotsProvider.getBundleStats( "0x123456789abcdef123456789abcdef123456789abcdef123456789abcdef1234", 13509887, ), );
```

Output:

```
json { "isSimulated": true, "isSentToMiners": true, "isHighPriority": true, "simulatedAt": "2021-10-29T04:00:50.526Z", "submittedAt": "2021-10-29T04:00:50.472Z", "sentToMinersAt": "2021-10-29T04:00:50.546Z" }
```

:::note

Parameters with `miner` in the name are retrofitted with Flashbots block builder data to maintain backwards compatibility. This nomenclature will be changed in a future release to accurately reflect PoS Ethereum architecture.

:::

Analyze the timestamps above in relation to when you observe the targeted block being propagated to your node.

- If the time difference is minimal, focus on reducing your processing time and network latency.
- If there is a significant delay between `sentToMinersAt` and when you observe the target block, proceed to the next section for further troubleshooting.

Is the validator for a particular block/slot running mev-boost?

5. A validator for target slot not running mev-boost

mev-boost is an opt-in system that runs alongside a validator's consensus client. Unless every validator on Ethereum runs mev-boost, some slots cannot be targeted with Flashbots. Additionally, your builder must be connected to the same relay as the proposer for the target slot.

If no bundles are detected in the blocks-api response, check if other blocks from the same validator ever have Flashbots bundles. If no blocks from a particular validator contain Flashbots bundles, it is possible your bundle was not seen by the validator who proposed the block for the target block height.

New Blocks API fields

New fields have been added to the blocks API to align the API with PoS Ethereum nomenclature. Details can be found in the Blocks API page.

Everything checks out, what's next?

Once you have validated the above issues are not affecting your bundle submission, consider filling out the Flashbots searcher support form:

Fill out our Searcher Issue Reporting Form

Be sure to include the output from the above RPC calls:

- eth_callBundle / simulate
- eth_getBundleStats
- getConflictingBundle

in the form submission.